Міністерство освіти та науки молоді і спорту України
Львівський національний університет імені Івана Франка

На правах рукопису

Волощук Орест Романович

УДК 004.853+004.855.5

Методи кластеризації на великих масивах даних

01.05.03 — математичне та програмне забезпечення обчислювальних машин і систем

Магістерська робота

Наукові керівники:
Притула Микола Миколайович,
проффесор,
Годич Олесь Васильович,
кандидат фізико-математичних наук, доцент

Львів — 2011

ЗМІСТ

ВСТУП

Актуальність теми. Сьогодні все частіше виникають задачі, так чи інакше пов'язані із розбиттям масиву об'єктів на групи за певними критеріями. Із розвитком обчислювальної техніки збільшуються також об'єми баз даних, що можуть піддаватись такому аналізу. Виникає необхідність створення ефективних алгоритмів опрацювання великих масивів даних.

Мета і завдання дослідження. Метою дослідження є оптимізація алгоритмів кластеризації для обробки великих масивів даних. Для досягнення цієї мети були сформульовані та вирішені такі основні завдання:

— провести порівняльний аналіз алгоритмів кластеризації
— розробити методи та виявити покращення, що дозволять прискорити виконання задачі кластеризації
— здійснити оцінку покращення роботи алгоритмів кластеризації за рахунок здійсненої оптимізації

Об'єкт дослідження. Об'єктом дослідження є процеси ієрархічної та роздільної кластеризації даних великих об'ємів.

Предмет дослідження. Предметом дослідження є оптимізація алгоритмів кластеризації для ефективного аналізу даних великих об'ємів.

РОЗДІЛ 1
ОГЛЯД СТАНУ ПРОБЛЕМИ ТА ОСНОВНІ ПОНЯТТЯ

1.1. Розвиток та застосування кластеризації

Термін „кластеризація", вперше використаний Тріоном у [17], означає набір підходів та алгоритмів, призначених для об'єднання схожих об'єктів у групи. Ця технологія знайшла своє застосування в цілій низці галузей наук та є необхідною частиною більшості сучасних засобів видобування знань.

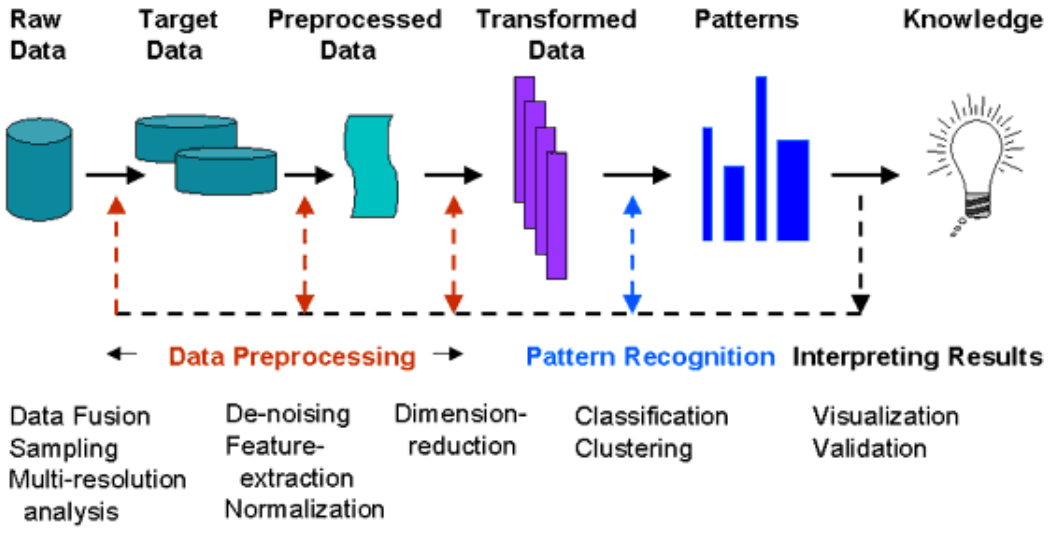

Рис. 1.1. Етапи видобування даних

Рис. 1.1 ілюструє етапи процесу видобування знань. Одним із його етапів є аналіз шаблонів та прихованих закономірностей в базах даних. Саме для пошуку таких закономірностей розроблено алгоритми кластеризації.

В 1959 радянський вчений Терентьєв розробив так званий „метод кореляційних плеяд" [19], призначений для групування об'єктів за їх корелюючими ознаками. Займаючись вивченням кореляцій між різними ознаками

озерної жаби, він об'єднав їх в групи за абсолютною величиною коефіцієнту кореляції. Таким чином він отримав дві групи ознак – ознаки із великим та з малим значенням кореляції. Терентьєв назвав ці групи „кореляційними плеядами" та опублікував декілька методів їх аналізу. Це посприяло розвитку методів кластеризації за допомогою графів.

На початку 50х років також вийшли публікації Р. Льюїса, Е. Фікса та Дж. Ходжеса, присвячені ієрархічним алгоритмам кластеризації. Відчутний поштовх технології кластерного аналізу дали роботи Розенблатта про розпізнаючий пристій „перцептрон" [11]. Ці роботи поклали початок теорії „розпізнавання без вчителя". Поштовхом до розробки методів кластеризації стала публікація [13]. В своїй роботі автори Сокел та Сніт виходили з того, що для створення ефективних біологічних класифікацій процедура кластеризації повинна використовувати всеможливі показники, що характеризують досліджувані організми, проводити оцінку ступеня схожості між цими організмами, та забезпечувати розташування схожих організмів в одну групу. При цьому сформовані групи повинні бути досить локальними, тобто схожість організмів всередині групи повинна бути більшою, ніж схожість між організмами, що належать різним групам. Подальший аналіз таких груп допоможе вияснити, чи відповідають вони реальним біологічним класифікаціям. Сокел та Сніт вважали, що виявлення структури розподілу об'єктів у групи допоможе встановити процес утворення цих груп.

В середині сімдесятих з'явилась також низка робіт, присвячених методам аналізу якості здійсненої кластеризації. Індекси Данна [5] та Девіса-Боулдіна [3] дозволяють чисельно оцінити якість розбиття даних на кластери.

1.2. Типові задачі

Результатом роботи алгоритмів кластеризації є певне розбиття множини вхідних даних на групи кластери. З кожного із цих кластерів можна виділити типових представників. Це дасть можливість надалі абстрагуватись від великого масиву даних, і при подальшій роботі із таким набором даних працювати не з кожним об'єктом кластера, а лише з таким типовим представником. Такий підхід дозволяє суттєво спростити аналіз даних. Кластеризація часто стає складовою якоїсь більшої задачі, інструментом підготовки даних для її розв'язання. Такі задачі завжди пов'язані із пошуком та виділенням змістовних структур із великих масивів даних.

До них можна віднести задачі сегментування та розпізнавання зображень, мовлення, пошуку прихованих закономірностей в даних. На практиці такі задачі виникають при розв'язуванні проблем, що виникають в медицині, соціології, економіці та низці інших сфер діяльності людини. У медицині, наприклад, техніка сегментування зображення дозволяє виділяти на томограмах окремі області і на підставі їх форми та забарвлення приймати ставити діагноз. В біології використовуються техніки кластеризації для виявлення взаємопов'язаних груп генів та їх впливу на живі організми. Кластеризація також успішно застосовується у маркетингових дослідженнях для виявлення зв'язків між різними групами споживачів та потенційних покупців, цільових аудиторій, та для оптимального позиціонування нової продукції. В соціологічних дослідженнях використовується кластеризація даних, отриманих з різних джерел, для спрощення їх подальшого аналізу, виділення закономірностей та груп населення.

У своїй праці [18] Загоруйко описує одну із таких задач. Новосибірські вчені вивчали причини переселення людей з сіл в міста. Були вислані експедиції в навколишні села, жителям яких задавали приблизно сто анкетних питань, що стосувались віку, сімейного становища, освіти та ін. Після за-

вершення опитування дослідники постали перед необхідністю аналізувати більш ніж сім тисяч анкет, що містили понад сто питань кожна. Ці дані було введено в програму таксономії, котра повернула сім великих таксонів, середні характеристики яких дозволили дати зібраним даним змістовну інтерпретацію. Наприклад, виділився кластер, що містив переважно жінок середнього віку, котрі мали дорослих дітей в місті. Очевидно, представниці цього таксону, названого дослідниками „бабусі", їхали в місто доглядати за своїми внуками. Інші таксони отримали свої назви відповідно до їх типових представників.

Перед процедурою кластеризації часто дані буває необхідно підготувати. В практиці нормою є випадки, коли для деяких об'єктів бракує частини атрибутів — в такому разі перед кластеризацією необхідно здійснити передбачення цих атрибутів, користуючись наявною інформацією. Також значення атрибутів об'єктів необхідно нормувати.

1.3. Основні поняття та означення

Дамо математичне означення кластеризації. Нехай D — множина точок n-вимірного простору.

Означення 1.1. *Кластеризацією* $C = \{C \mid C \subseteq D\}$ називається таке розбиття D на підмножини, для якого виконується $\cup_{C_i \in C} = D$ і $\forall C_i, C_j \in C : C_i \cap C_{j \neq i} = \varnothing$. Множини C_i називаються кластерами.

Як вже згадувалось раніше, задача кластеризації полягає в знаходженні такого розбиття C, щоб схожі між собою об'єкти належали до одного кластера, а не схожі — до різних. Необхідно визначити спосіб обчислення схожості об'єктів. Для цього на просторі об'єктів вводиться певна метрика, геометричний зміст якої — відстань між об'єктами. Вона використовується як величина, обернена до міри схожості між об'єктами. На даний момент розроблено і широко застосовується наступний набір метрик:

— евклідова відстань

$$\rho(x, x') = \sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2} \tag{1.1}$$

— квадрат евклідової відстані

$$\rho(x, x') = \sum_{i=1}^{n}(x_i - x'_i) \tag{1.2}$$

(використовується для надання більшої ваги об'єктам, розташованим далеко один від одного)

— манхеттенська відстань

$$\rho(x, x') = \sum_{i=1}^{n} \mid x_i - x'_i \mid \tag{1.3}$$

— відстань Чебишева

$$\rho(x, x') = max(\mid x_i - x'_i \mid) \tag{1.4}$$

ця метрика дозволяє розрізнити об'єкти, якщо вони відрізняються лише одною координатою

1.4. Алгоритми кластеризації

На даний момент значного розвитку набула ціла низка різноманітних алгоритмів кластеризації. Їх можна розділити на групи за різними ознаками:

— спосіб групування (роздільне, ієрархічне)

— визначеність

— чутливість до форми кластерів

Роздільні алгоритми будують одне розбиття вибірки на кластери. На відміну від роздільних, агломеративні будують цілу систему взаємовкладених кластерів. На виході алгоритму отримується дерево кластерів, коренем якого служить кластер, що містить усі об'єкти вибірки, а листками є найменші кластери, кожен з яких містить по одному об'єкту. В такому разі дослідник має можливість обрати такий переріз дерева кластерів, що найкраще відповідатиме його потребам.

Визначені алгоритми однозначно ставлять у відповідність кожному об'єкту один кластер. Невизначені не дають такої чіткої інформації. Замість ідентифікатора кластера вони повертають імовірність, із якою об'єкт належить до кожного із кластерів.

На сьогоднішній день широко використовуються наступні алгоритми кластеризації:

— k-means [16, 8]
— UPGMA [14]
— DBSCAN [4]
— FOREL [18]
— c-means
— карти Кохонена [11]

1.5. Висновки до розділу 1

Нині набула поширення низка практичних задач, розв'язання яких вимагає виявлення прихованих закономірностей у наборі даних. Задачі такого типу виникають у медицині, соціології, економіці та низці інших галузей людської діяльності. Кластеризація набула великого значення як засіб розв'язання таких задач. У зв'язку із розвитком інформаційних технологій спостерігається значне зростання об'ємів даних, що піддаються збору та обробці. Така тенденція вимагає створення ефективних алгоритмів кла-

стеризації, здатних обробляти масиви даних великого об'єму.

У першому розділі наведено відомості про розвиток алгоритмів кластеризації до сьогоднішнього дня. Також у цьому розділі подано означення основних понять, пов'язаних із задачами кластеризації, та формальне означення самого поняття кластеризації. У 1.3 описано популярні алгоритми кластеризації та їх розбиття на групи за кількома критеріями.

Для здійснення кластеризації необхідно визначити метрику простору, міру відстані між об'єктами вибірки. В розділі наведено декілька основних та найпопулярніших метрик, що нині використовуються для кластеризації.

РОЗДІЛ 2
АНАЛІЗ АЛГОРИТМІВ

В даній роботі розглядається ефективність чотирьох алгоритмів кластеризації. Два із них здійснюють роздільну кластеризацію: k-means та DBSCAN. Інші два — ієрархічну: UPGMA та споріднений із ним Neighbor-joining.

2.1. Опис алгоритмів

K-Means. K-means — це метод кластеризації, що повертає таке розбиття, в якому кожен об'єкт належить кластеру із найближчим центром. Задача побудови такого розбиття належить до класу NP [9], тому на практиці застосовуються евристичні реалізації. Для роботи алгоритму потрібно заздалегідь володіти інформацією про кількість кластерів, на які розбивається вибірка.

Оскільки така реалізація є евристичною, результатом її роботи може стати не глобальне, а локальне розбиття. Зокрема, вони залежать від вибору первинних центроїд кластерів на етапі ініціалізації. До цього моменту є декілька відомих підходів, описаних зокрема в [7]. Можливим варіантом є вибір у якості центроїд випадкових об'єктів із вибірки. Інший варіант — призначення кожного об'єкта у випадковий кластер та вибір центрів цих кластерів як початкових центроїд.

Стандартною практикою при застосуванні k-means є кількаразовий запуск алгоритму із різними вхідними даними та вибір того результату, який повторюється найчастіше.

Для деяких наборів даних k-means збігається за експоненціальний час

Алгоритм 1 Алгоритм k-means

Ініціалізація. Виберемо у довільний спосіб k точок $c_1^{(1)}, c_2^{(1)}, ..., c_k^{(1)}$, що стануть центроїдами кластерів. Номер ітерації $t := 1$.

1. Заповнюємо кожен із k кластерів тими об'єктами, для яких його центроїда є найближчою з-поміж центроїд усіх кластерів:

$$S_i^{(t)} = \{x_j : ||x_j - c_i^{(t)}|| \leqslant ||x_j - c_{c^*}^{(t)}||, i^* = 1, 2, ..., k\} \tag{2.1}$$

2. Порівняємо розподіл об'єктів по кластерах із отриманим на попередньому кроці. Якщо вони збігаються, алгоритм завершено. В іншому разі, переходимо до кроку 3.
3. Обчислимо нові центроїди кластерів $c_1^{(t+1)}, c_2^{(t+1)}, ..., c_k^{(t+1)}$:

$$c_i^{(t+1)} = \frac{1}{S_i^{(t)}} \sum_{x_j \in S_i^{(t)}} x_j \tag{2.2}$$

Збільшуємо номер ітерації: $t := t + 1$. Переходимо до кроку 1.

[2], але такі набори на практиці не зустрічаються. Згладжена складність виконання такого алгоритму — поліноміальна [1].

Цей метод має суттєві недоліки. Один із найважливіших — необхідність задавати кількість кластерів перед запуском. Неправильно обрана кількість кластерів може призвести до некоректного розбиття. Тому після кожного виконання алгоритму потрібно проводити діагностичні перевірки. Крім того, алгоритм розрахований лише на кластери сферичної форми та приблизно однакового розміру. При аналізі вибірки, що містить одне велике скупчення та декілька маленьких, k-means схильний розколювати велике скупчення.

UPGMA. Unweighted Pair Group Method with Arithmetic Mean — алгоритм ієрархічної кластеризації, призначений для побудови філогенетичних дерев. Такі дерева відображають еволюційні взаємозв'язки між різними видами або іншими сутностями, що мають спільного предка.

на початку роботи алгоритму створюється набір кластерів, кожен із

Алгоритм 2 Алгоритм UPGMA

Ініціалізація Створимо масив кластерів D, кожен із яких міститиме по одному об'єкту із вибірки.

1. Якщо кількість кластерів рівна одиниці, алгоритм завершено. Інакше, переходимо до кроку 2.
2. Знайдемо пару кластерів (A, B) таку, що

$$d(A, B) = \min_{A,B \in D; A \neq B} d(A, B) \tag{2.3}$$

де

$$d(A, B) = \frac{1}{\mid A \mid\mid B \mid} \sum_{x \in A} \sum_{y \in B} d(x, y) \tag{2.4}$$

3. Об'єднаємо пару A, B у кластер C. Вилучимо A, B із D, та додамо C у D.

яких містить по одному об'єкту вибірки. На кожному кроці здйснюється об'єднання двох найближчих кластерів у один. За відстань між кластерами береться середня відстань між усіма парами їх об'єктів, тобто:

$$d(A, B) = \frac{1}{\mid A \mid\mid B \mid} \sum_{x \in A} \sum_{y \in b} d(x, y) \tag{2.5}$$

Після завершення роботи алгоритму отримується один кластер, який містить усі об'єкти вибірки. Процес об'єднання кластерів під час роботи алгоритму можна зобразити у вигляді дерева. Воно ілюструє зв'язки між кластерами.

UPGMA належить до групи алгоритмів GCP (Global Closest Pair). На кожній ітерації необхідно знайти пару кластерів, найближчих одне до одного. Для цього необхідно мати доступ до матриці відстаней між усіма об'єктами. Очевидно, для досить великих масивів даних обчислення такої матриці займає багато часу, а кешування її потребує великих затрат пам'яті. Використання спеціальної структури даних для зберігання відстаней між об'єктами дозволило реалізувати алгоритм зі складністю $O(n^2)$ [6] відносно часу, що є очевидною нижньою межею для цього класу алгоритмів.

DBSCAN. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [4] — алгоритм роздільної кластеризації, розроблений у 1996. Це один із найчастіше згадуваних та використовуваних алгоритмів. DBSCAN здійснює розбиття, грунтуючись на густині розташування об'єктів вибірки, а не на відстані до центру. Це дозволяє виділяти кластери довільної форми. Алгоритм розрахований на роботу із даними, що містять певну кількість шуму. Деякі об'єкти вибірки можуть залишились без кластера. Алгоритм не потребує інформації про кількість кластерів.

DBSCAN приймає на вхід два параметри — ϵ (окіл, в якому слід шукати сусідів точки) та n (мінімальна кількість сусідів, необхідна для того, щоб точку не визнали шумом). DBSCAN аналізує усі точки вибірки. Ті з них, в ϵ-околі яких є більш ніж n точок, додаються у кластери. Решта визнаються шумом.

Алгоритм 3 Алгоритм DBSCAN

1. Виберемо довільну точку p серед досі не відвіданих. Позначимо p як відвідану.
2. Знайдемо всіх її сусідів, що лежать в межах ϵ-околу p:
$$N = \{x : d(p, x) < \epsilon\} \tag{2.6}$$
3. Якщо $\mid N \mid > n$, створюємо новий кластер, додаємо до нього p. Інакше переходимо до кроку 1.
4. Вибираємо довільну точку p' серед невідвіданих точок із N. Позначаємо її як відвідану.
5. Шукаємо всіх її сусідів:
$$N' = \{x : d(p', x) < \epsilon\} \tag{2.7}$$
6. Якщо $\mid N' \mid > n$, додаємо p' до кластера та доповнюємо N:
$$N = N \cup N' \tag{2.8}$$
7. Якщо у N залишились невідвідані точки, переходимо до кроку 4, інакше закриваємо кластер та переходимо до кроку 1.

DBSCAN на кожній ітерації шукає найближчих сусідів одної з точок

вибірки. При прямій реалізації, що передбачає обчислення усього рядка матриці відстаней на кожній ітерації, загальна складність алгоритму становить $O(n^2)$. При використанні індексованої структури даних для збереження матриці відстаней (наприклад, R-дерева, яке дозволяє виконати пошук сусідів в межах деякого околу за $(O(logn)))$, загальна складність алгоритму становитиме $O(nlogn)$. При використанні неоптимізованої структури даних складність становитиме $O(n^2)$. Щоправда, потреби у пам'яті також виростуть до $O(n^2)$.

Neighbor-joining. [12] Це алгоритм ієрархічної кластеризації. Він споріднений із UPGMA, але між ними є суттєві відмінності. Якщо у випадку UPGMA на кожній ітерації ми шукаємо глобально найближчу пару кластерів, то при neighbor joining здіснюється пошук та об'єднання кластерів, які є локально найближчими сусідами одне одного. Для цього достатньо зберігати повний ланцюжок найближчих сусідів. Такий підхід дає можливість вагомо зменшити кількість обчислень. Складність алгоритму визначатиметься кількістю пошуків найближчих сусідів.

Означення 2.1. Ланцюгом найближчих сусідів називається такий кортеж $P = (C_1, C_2, ..., C_N)$, в якому C_{i+1} є найближчим сусідом C_i.

Означення 2.2. Ланцюг найближчих сусідів називається повним, якщо він містить хоча б два кластери, і два останні кластери є взаємно найближчими сусідами.

На початку роботи алгоритму вибірка розбивається на кластери за правилом "один об'єкт — один кластер". Після цього будується повний ланцюг найближчих сусідів. Після цього останні два кластери об'єднуються в один і вилучаються із ланцюга найближчих сусідів, і побудова повного ланцюга найближчих сусідів продовжується. Якщо ланцюг спорожнів, починаємо будувати його знову із довільним чином вибраного кластера.

Додавання одного кластера до ланцюга і перевірка повноти ланцюга має

Алгоритм 4 Алгоритм Neighbor-joining

Ініціалізація Створимо множину одинарних кластерів із усіх об'єктів вибірки.

1. Виберемо будь-який кластер p.
2. Побудуємо повний ланцюжок його найближчих сусідів.
3. Об'єднаємо два останні кластери у ланцюжку в один та вилучимо їх із ланцюжка.
4. Якщо в ланцюжку залишились елементи, продовжимо побудову із останнього елемента та переходимо до кроку 3. Інакше, переходимо до наступного кроку.
5. Якщо кількість кластерів більша, ніж 2, переходимо до кроку 1. Інакше, об'єднуємо ці кластери в один. Алгоритм завершено.

часову складність $O(n)$. Ця операція здійснюється не більш ніж $2(n-1)$ разів. Таким чином, загальна часова складність алгоритму дорівнює $O(n^2)$.

2.2. Висновки до розділу 2

У розділі 2 описано чотири алгоритми кластеризації: k-means, DBSCAN, UPGMA та neighbor-joining. K-means та DBSCAN належать до класу алгоритмів роздільної кластеризації, а UPGMA та Neighbor-joining здійснюють ієрархічну кластеризацію.

Кожен із цих алгоритмів має власні недоліки та переваги. Вони стосуються швидкості роботи, вимог до кількості пам'яті, чутливості до форми та розмірів кластерів, здатності розпізнавати та ігнорувати шум в даних. K-means та neighbor-joining — евристичні алгоритми. Це означає, що результат роботи цих двох алгоритмів не обов'язково буде оптимальним. Вказано оцінку складності алгоритмів та їх потреби у пам'яті. Вказано особливості алгоритмів, їх переваги. Наведено ситуації, в яких найдоцільніше використати конкретний алгоритм.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ АЛГОРИТМІВ

3.1. Тестові дані

Для перевірки алгоритмів використовувались синтетичні набори даних. Була написана програма, що створює набір даних заданої форми. Утиліта зчитує із вказаного файла послідовно інформацію про центр групи об'єктів і дозволені відхилення від центру по кожній координаті та генерує вказану кількість об'єктів, розташованих випадковим чином у вказаних межах навколо центра. Задавши достатню кількість центральних точок і вибравши відповідні відхилення, можна отримати набір даних довільної форми. Зокрема, задавши центр групи приблизно посередині між усіма іншими точками, і досить великі відхилення по усіх координатах, можна досягнути ефекту шуму в даних ("зоряне небо"). Як і основна програма, утиліта написана на C++.

3.2. Програмне забезпечення

Для перевірки ефективності вищеописаних алгоритмів була створена їх програмна реалізація. У виборі мови програмування я керувався міркуваннями швидкодії та можливості контролю ресурсів комп'ютера. Вибір зупинився на C++. Реалізація являє собою консольну програму без графічного інтерфейсу, що зчитує дані з файла та зберігає у файлі результати кластеризації.

Створено об'єктно-орієнтовану модель для зберігання об'єктів вибірки. Для зберігання власне даних використовуються стандартні колекції STL. Де-юре STL не належить до стандарту мови C++, але де-факто усі суча-

сні поширені компілятори мають підтримку цеї бібліотеки. Перша версія програми включала об'єкт „DataContainer", який містив асоціативний контейнер std::map, що дозволяв звертатись до обєктів вибірки по ідентифікатору. Кожен елемент вибірки був екземпляром класу Object, що містив std::vector вказівників на об'єкти класу Attribute. Кожен екземпляр Attribute містив числове значення відповідного атрибута та інформацію при присутність даного атрибута у відповідного об'єкта вибірки.

Реалізовано також абстрактний клас AbstractMetric, який відповідав метриці простору. Архітектура програми дозволяє реалізувати довільну метрику простору. Таким чином при реалізації власне алгоритму кластеризації стало можливим абстрагуватись від типу метрики за допомогою механізму поліморфізму, доступного у C++. На даний момент із метрик реалізовано найпопулярнішу евклідову метрику.

Написання програми відбувалось у текстовому редакторі vim. Для компіляції програми обрано компілятор gcc. Відлагодження програми відбувалось за допомогою набору інструментів gdb. Відслідковування витрат пам'яті та профілювання програми здійснювалось за допомогою valgrind. При розробці використовувалась система контролю версій git.

Запуски програми та заміри часу проводились на комп'ютері із двоядерним процесором Intel Pentium D з тактовою частотою 2.8 ГГц. На комп'ютері встановлено 3 Гб оперативної пам'яті типу DDRII, що працює на частоті 667 МГц.

Компіляція виконувалась із вказанням -O3 в якості рівня автоматичної оптимізації, в конфігурації Release.

3.2.1. K-means. K-means — ітераційний алгоритм, тому час його роботи залежить не лише від об'єму вхідних даних, а і від їх структури. Тому оцінити загальний час роботи алгоритму складно. Натомість будемо оцінювати швидкодію конкретної реалізації за часом виконання одної ітерації.

На кожній ітерації k-means здійснюється обчислення центроїди кожного кластера, та обчислення відстані від кожного об'єкта до кожної центроїди. Оскільки на кожній ітерації кожен об'єкт повинен увійти в кластер, сумарна складність обчислення центроїд не змінюється між ітераціями. Кількість центроїд, очевидно, рівна кількості кластерів, тому час пошуку найближчої центроїди також залишається незмінним. Таким чином, час виконання ітерації на протязі усієї роботи алгоритму не змінюється. Це підтверджується графіком на рис. 3.1. Цей же графік ілюструє вплив розмірності простору та заданої кількості кластерів на час роботи ітерації. Розмірність простору впливає на складність обчислення відстані між об'єктами. на кожній ітерації для кожного об'єкта потрібно знайти найближчу центроїду. Кількість центроїд дорівнює кількості кластерів, тому остання прямо впливає на складність цього кроку.

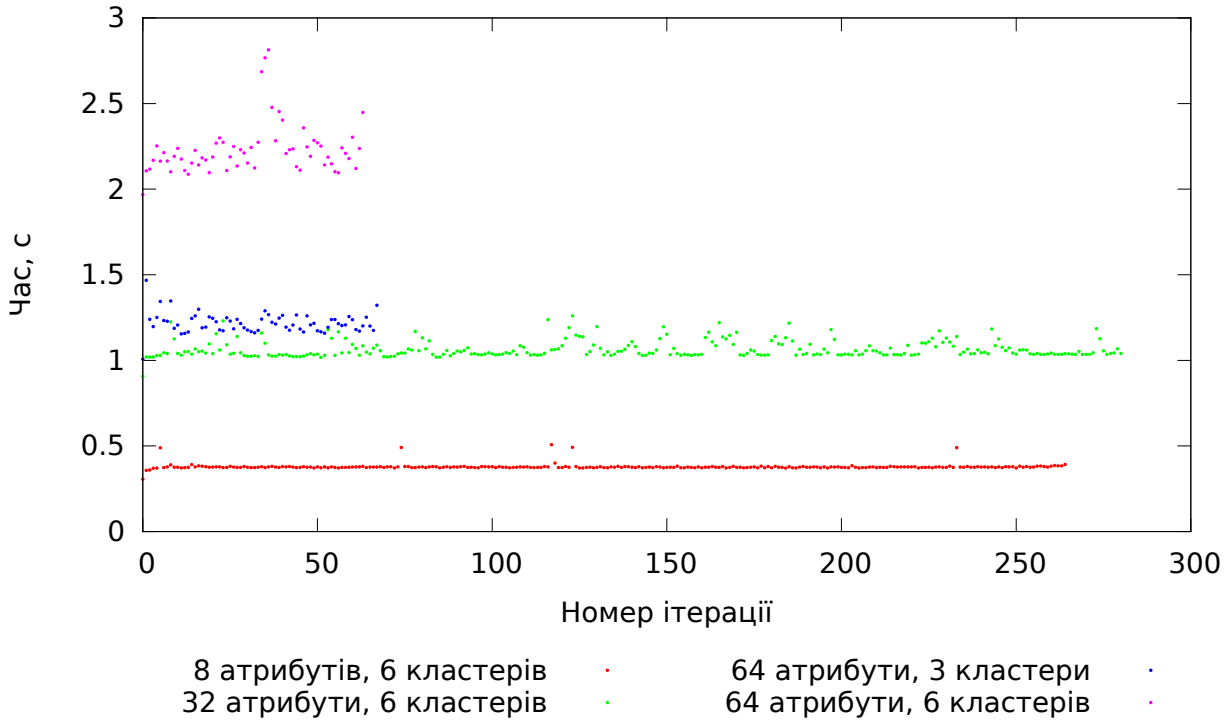

Рис. 3.1. Час виконання кожної ітерації k-means

На вхід алгоритму подавались три схожі синтетичні набори даних. Об'єкти в кожному з них розташовуються випадковим чином всередині n-

вимірного куба з центром в початку координат та довжиною сторони, що дорівнює 2. Кількість об'єктів у кожному наборі — 100000. Ці набори даних містили відповідно об'єкти з 8-вимірного, 32-вимірного та 64-вимірного просторів. Ці набори алгоритм повинен був розбити на 3 та 6 кластерів.

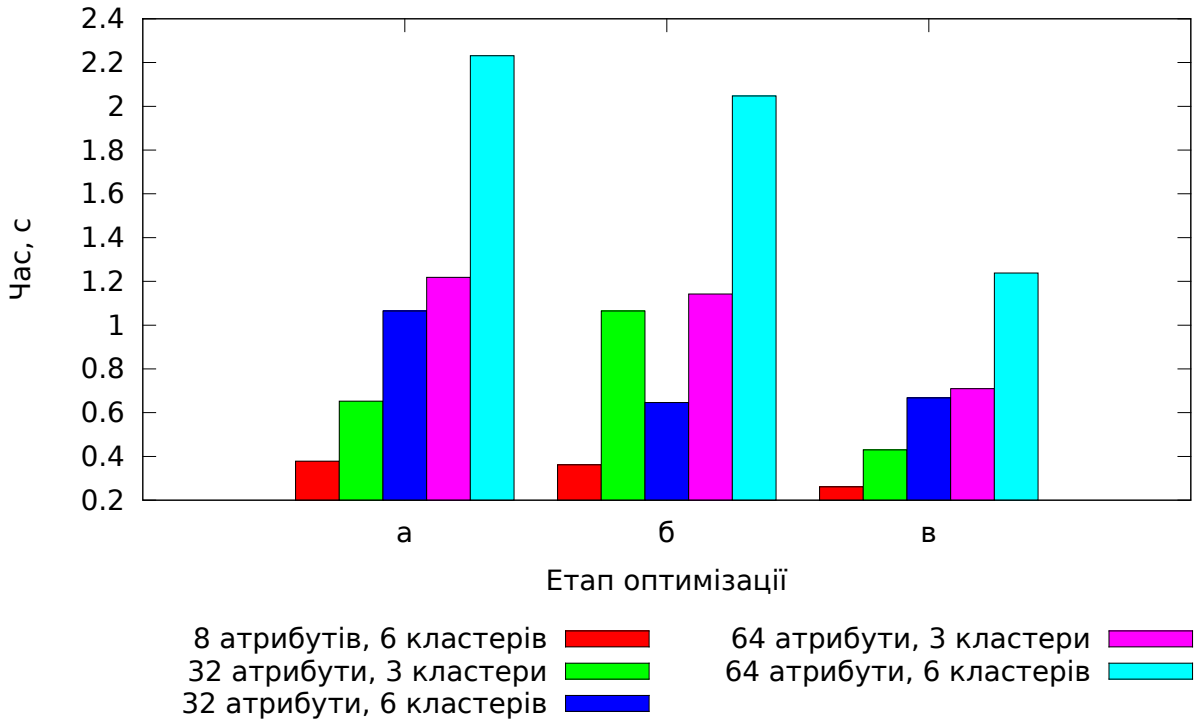

Рис. 3.2. Середній час виконання ітерацій k-means

Рис. 3.2 показує середній час виконання ітерації k-means для різних вхідних даних. Рис. 3.2, *а* ілюструє роботу k-means у первинному вигляді. Рис. 3.2, *б* показує час роботи алгоритму після переходу на числа одинарної точності. Як бачимо, ця зміна суттєво вплинула лише на варіант запуску, що вимагав найбільше обчислень.

Профілювання показало, що вибрана структура даних спричиняє великі втрати. За задумом, кожен екземпляр класу Object містив вектор екземплярів класу Attribute. Клас Attribute містив значення відповідного атрибуту та деяку додаткову інформацію (зокрема, дані про те, чи цей атрибут відомий із вхідного масиву). Це дозволяло легко розширити програму, додавши до неї функціонал передбачення даних, яких бракує у вибірці.

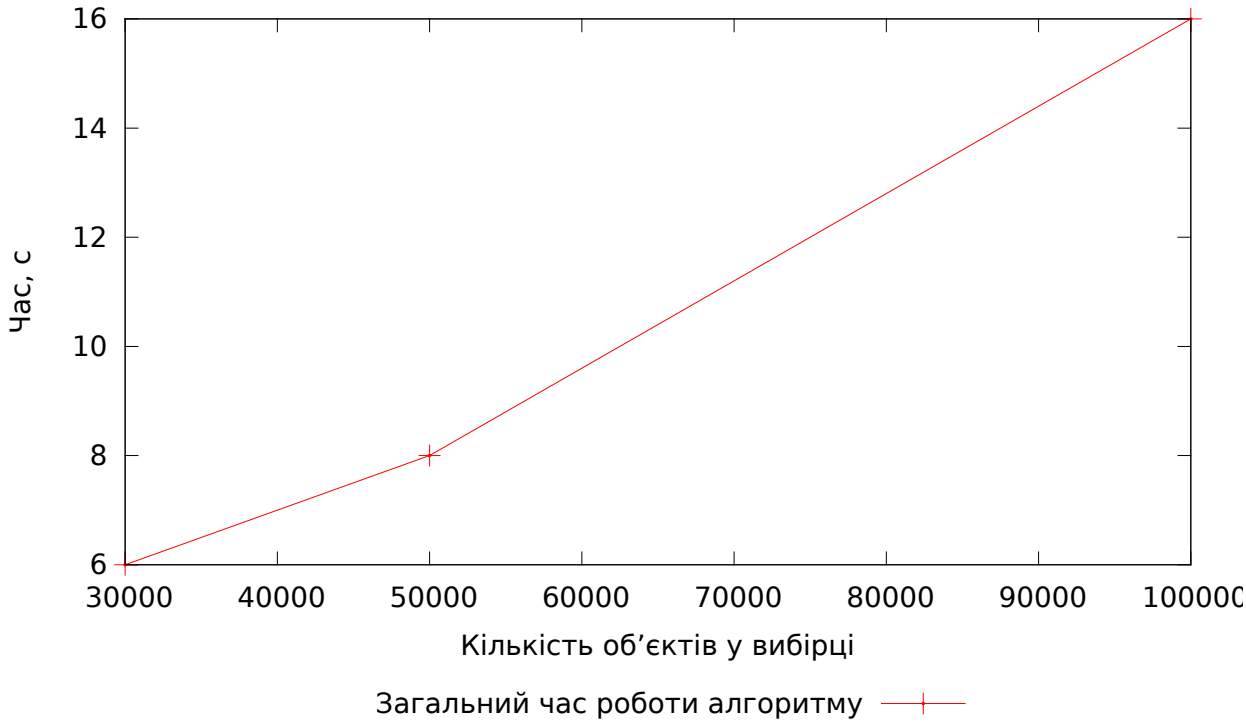

Рис. 3.3. Загальний час роботи алгоритму k-means

За результатами профілювання від цеї структури довелось відмовитись. Замість вектора об’єктів використано динамічний масив чисел. Швидкість роботи k-means у такому випадку показано на рис. 3.2, *в*. Можна бачити, що така модифікація відчутно зменшила час кожної ітерації.

На рис. 3.3 показано час роботи алгоритму k-means для вхідних даних різного об’єму.

3.3. DBSCAN

У DBSCAN кількість ітерацій рівна кількості об’єктів. На кожній ітерації вибирається один з об’єктів вибірки та обчислюються відстані від нього до усіх інших об’єктів вибірки. Таким чином, складність обчислень не змінюється між ітераціями. Проте, на відміну від k-means, залежно від результатів обчислень в ітерації можуть проводитись додаткові дії (такі як, наприклад, додавання сусідів об’єкта до списку об’єктів для подаль-

шого аналізу). За рахунок додаткових операцій із пам'яттю ця особливість впливає на час роботи алгоритму. Відхилення від середнього значення часу виконання ітерації спостерігались в обидва боки (в менший бік, якщо в об'єкта немає достатньої кількості сусідів, і ніяк не впливає на список об'єктів для розгляду, і в більший, якщо об'єкт додає своїх сусідів до списку розгляду). Максимальний розмір відхилення — 20% від часу виконання ітерації.

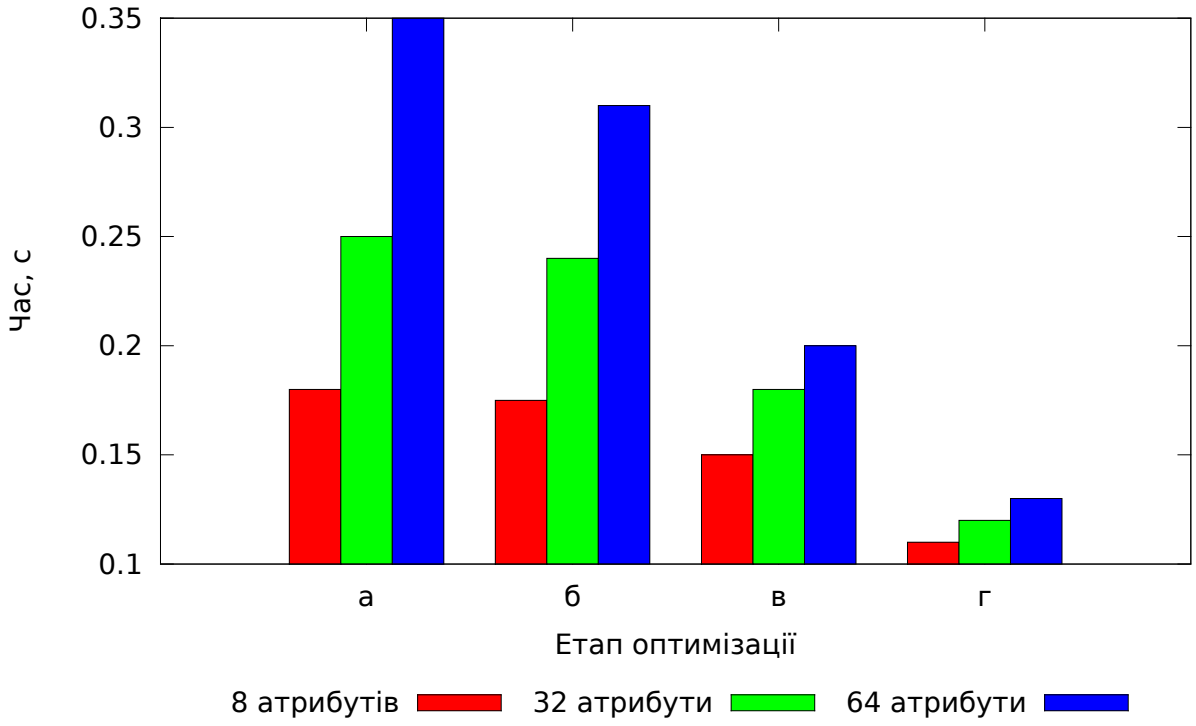

Рис. 3.4. Середній час виконання ітерацій DBSCAN

На вхід алгоритму подавався той самий масив даних, що і для k-means. На рис. 3.4 вказано середній час ітерації для кожного проведеного етапу оптимізації. На рис. 3.4, *а* зображено час виконання одної ітерації для початкової версії алгоритму. Рис. 3.4, *б* показує час виконання одної ітерації після переходу від чисел із подвійною точністю до чисел одинарної точності. Як і у випадку k-means, цей перехід мав суттєвий вплив лише на час обробки даних великої розмірності. Наступна група, рис. 3.4, *в*, показує час виконання одної ітерації алгоритму після відмови від складної структури

доступу до атрибутів об'єктів. Знову, аналогічно до k-means, цей перехід суттєво прискорив роботу алгоритму. Важливо звернути увагу на рис. 3.4, *г*. Це — результат оптимізації, що полягає у покращенні роботи з пам'яттю. Ця зміна призвела до зникнення залежності між кількістю близьких сусідів в об'єкта, що розглядається на певній ітерації, та часом виконання цеї ітерації. Такого покращення вдалось досягнути, відмовившись від деяких фактично зайвих операцій з пам'яттю.

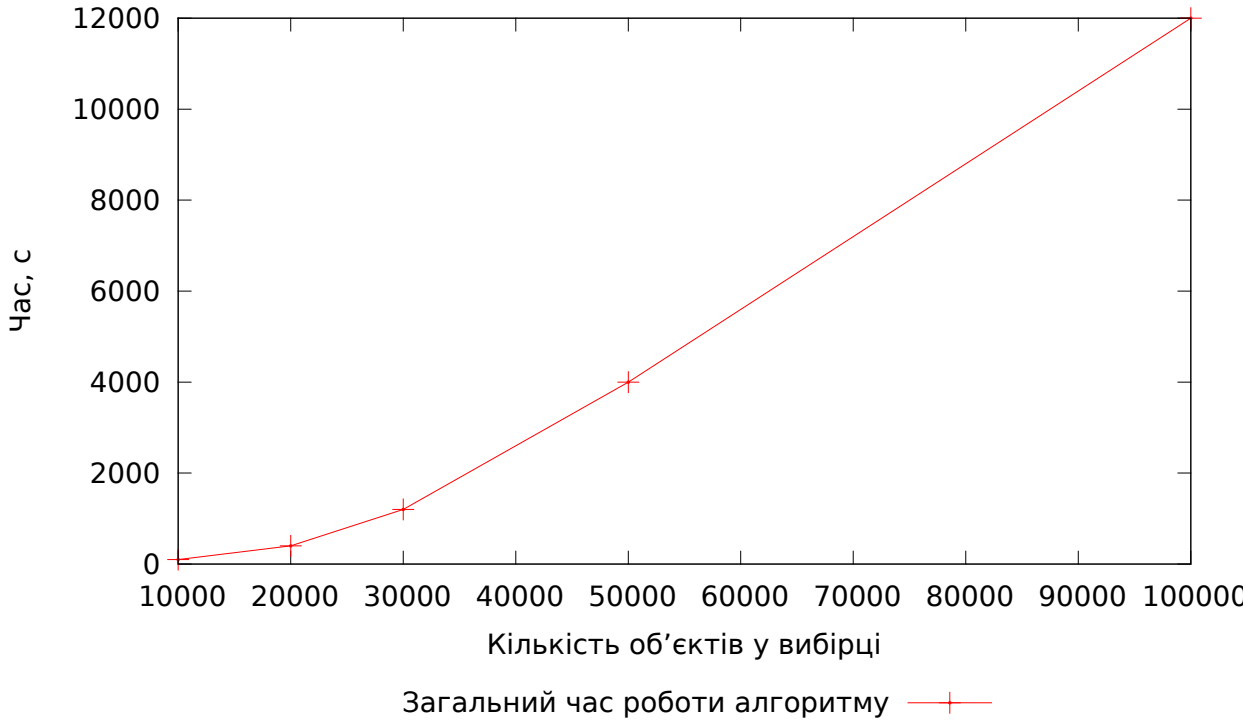

Рис. 3.5. Загальний час роботи DBSCAN

На рис. 3.5 бачимо залежність часу роботи реалізації DBSCAN від об'єму вхідних даних.

3.3.1. UPGMA. UPGMA на кожній ітерації потребує матриці відстаней між усіма об'єктами вибірки. Це залишає декілька можливих варіантів. Перший із них — обчислити матрицю відстаней на початку роботи алгоритму та зберігати її в оперативній пам'яті. Недолік цього підходу очевидний — для зберігання матриці відстаней між n об'єктами потрібно $\frac{n(n-1)}{2}$

одиниць пам'яті. Якщо для кожного числа виділити всього чотири байти, то загалом для збеігання матриці відстаней для масиву об'єктів розміром 100 000 потрібно буде $4\frac{100000*99999}{2} = 19999800000$ байт, або приблизно 19 терабайт.Такі об'єми пам'яті на даний момент доступні лише у великих обчислювальних центрах.

Другий варіант - використовувати спеціальну структуру даних, наприклад, R-дерево [10]. R-дерево призначене для пошуку розташованих поруч точок багатовимірного простору. Використання такої структури дозволить суттєво підвищити ефективність UPGMA. Але процедура побудови R-дерева обчислювально складна та зводиться до задачі кластеризації.

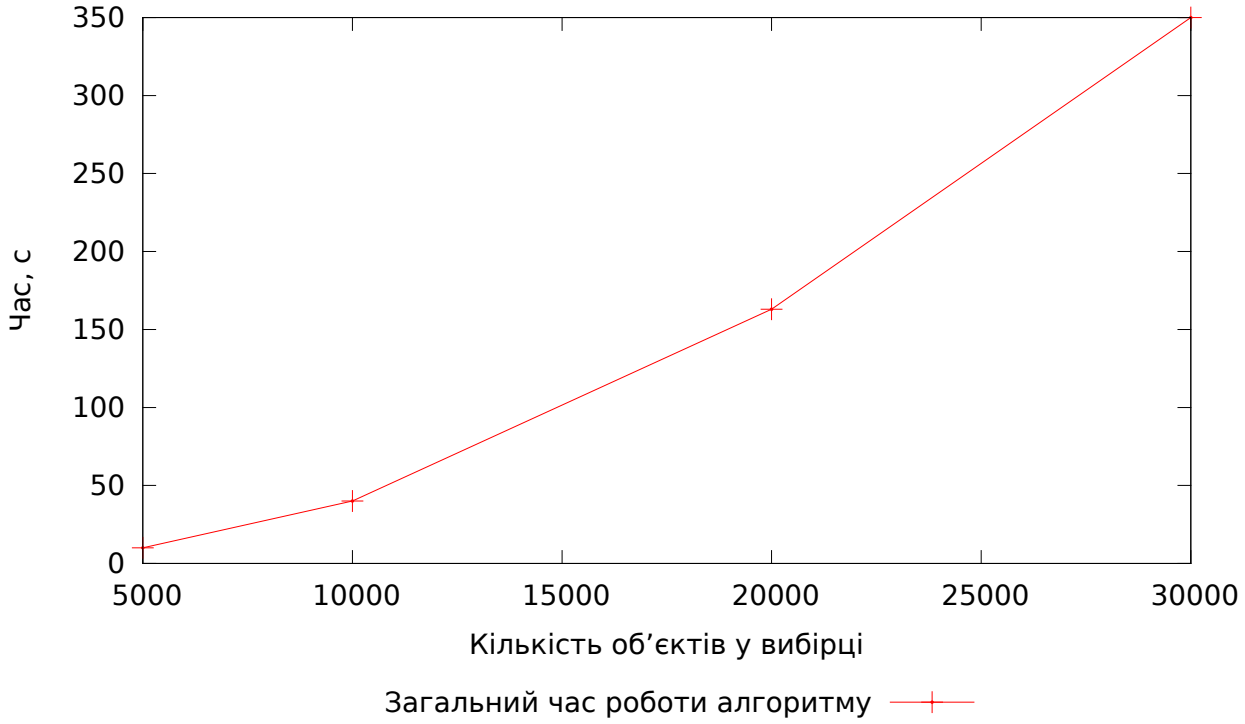

Рис. 3.6. Залежність часу виконання ітерації UPGMA від об'єму даних

Останній варіант — пряма реалізація UPGMA із обчисленням матриці відстаней на кожній ітерації. Це ліквідує надзвичайно великі вимоги до пам'яті, але надасть алгоритму складності $O(n^3)$.

Написана програма містить реалізацію UPGMA із обчисленням матриці відстаней на кожній ітерації. Результати оцінки часу показують, що при

задовільній тривалості виконання одної ітерації для масиву із 5000 об'єктів (10с), ситуація відчутно погіршується навіть при незначному збільшенні об'єму даних. Вже для 10000 об'єктів одна ітерація займає 40 секунд, а для 30000 — 350 секунд. Таке швидке зростання часу змушує визнати, що UPGMA за рахунок високої обчислювальної складності в чистому вигляді малопридатний для кластеризації великих об'ємів даних.

3.3.2. Neighbor-joining. Цей алгоритм — спрощення UPGMA. Для нього також точно відома кількість ітерацій, вона співпадає із кількістю ітерацій UPGMA та з кількістю об'єктів у вибірці. На кожному кроці два кластери об'єднуються в один. Різниця полягає лише в тому, яким чином здійснюється вибір кластерів для об'єднання.

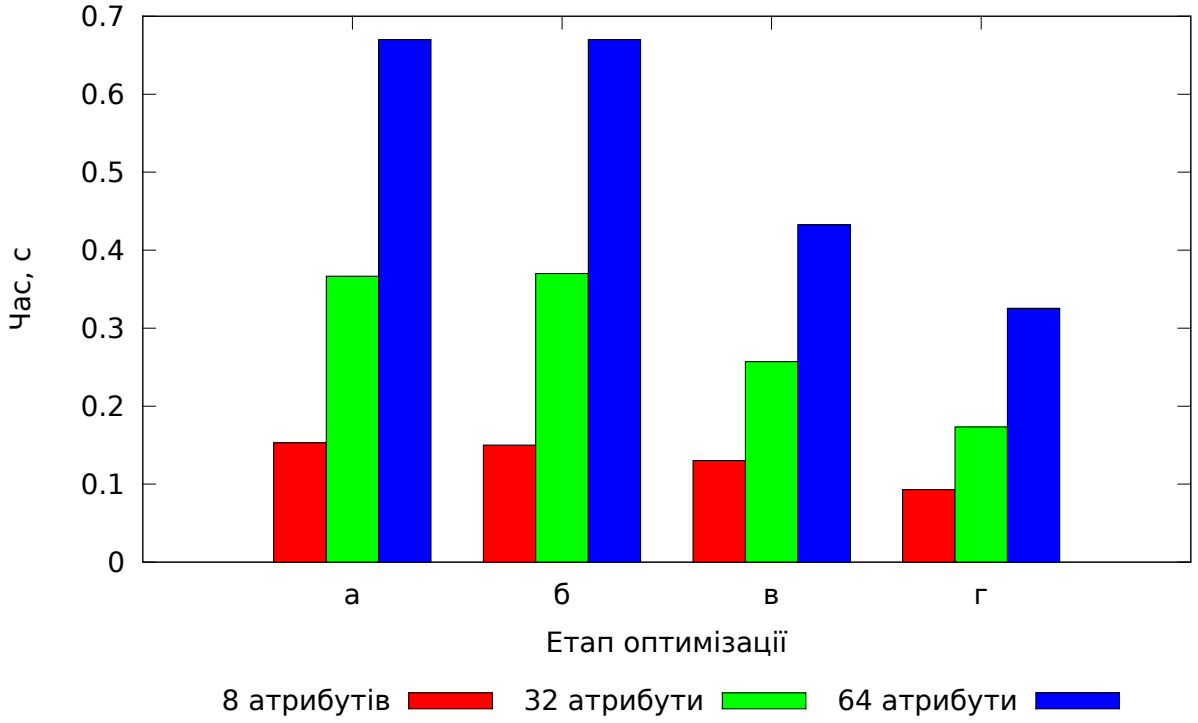

Рис. 3.7. Середній час виконання ітерацій neighbor-joining

На кожній ітерації алгоритм вибирає один із об'єктів вибірки та знаходить його найближчого сусіда.

Між ітераціями змінюється лише розмір вектора відстаней до класте-

рів, в якому ми шукаємо мінімум. Порівняно із основними обчисленнями в ітерації — пошуком відстаней від одного кластера, що розглядається на ітерації, до усіх інших — ця різниця є достатно незначною, щоб нею можна було знехтувати. Таким чином, загальну швидкодію алгоритму можна оцінити за середнім часом виконання одної ітерації.

Наведемо діаграму середнього часу виконання ітерацій алгоритму після введених покращень. На рис. 3.7, *а* показано час виконання ітерації в початковому варіанті реалізації. Рис. 3.7, *б* та *в* ілюструють вплив на швидкодію алгоритму переходу до чисел із одинарною точністю та зміни структури даних.

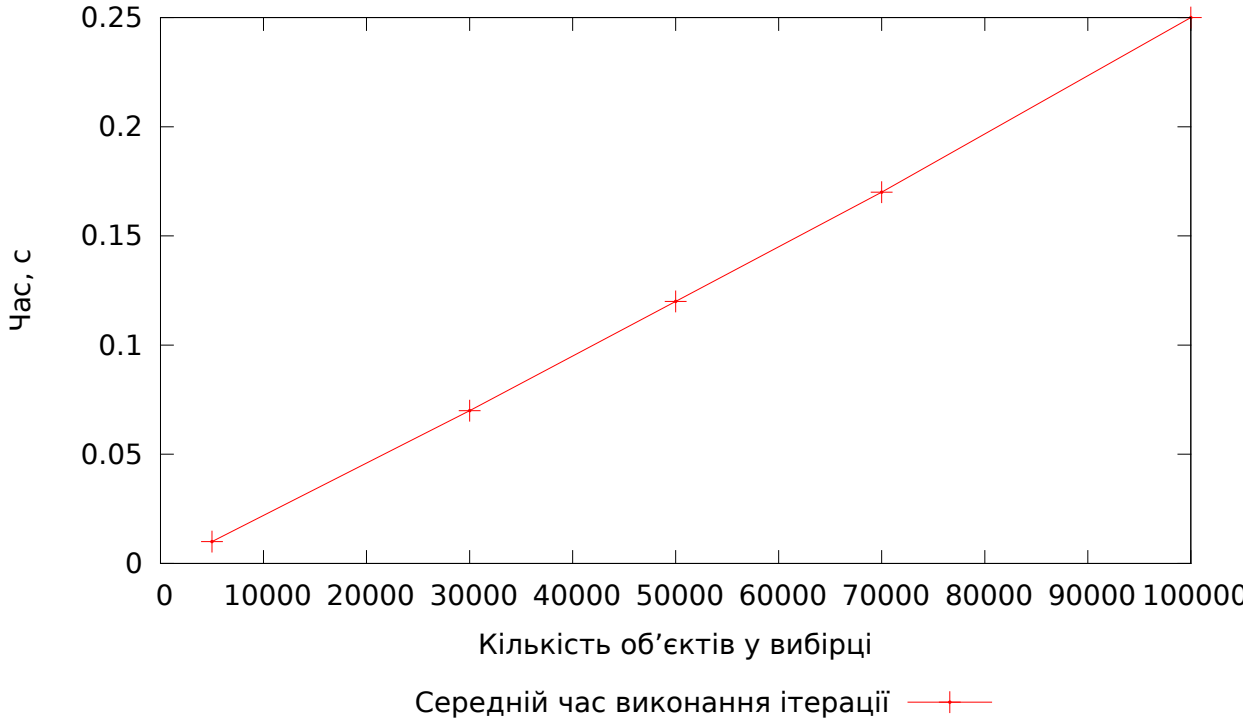

Рис. 3.8. Залежність часу виконання ітерації neighbor-joining від об’єму даних

Найбільший інтерес становить рис. 3.7, *г*. Для цього алгоритму реалізовано механізм багатопотокових обчислень. Структура алгоритму дала змогу виділити декілька окремих потоків виконання програми для обчислення відстаней між об’єктами вибірки та пошуку найближчого сусіда. Для бага-

тоядерних систем такий підхід дозволяє відчутно підвищити ефективність обчислень.

На рис. 3.8 показано середній час виконання ітерації neighbor-joining для різних розмірів вхідних даних. Легко бачити, що час ітерації лінійно залежить від розміру масиву об'єктів. Кількість ітерацій також зростає лінійно разом із об'ємом вхідних даних. Отже, в результаті отримано реалізацію алгоритму зі складністю $O(n^2)$.

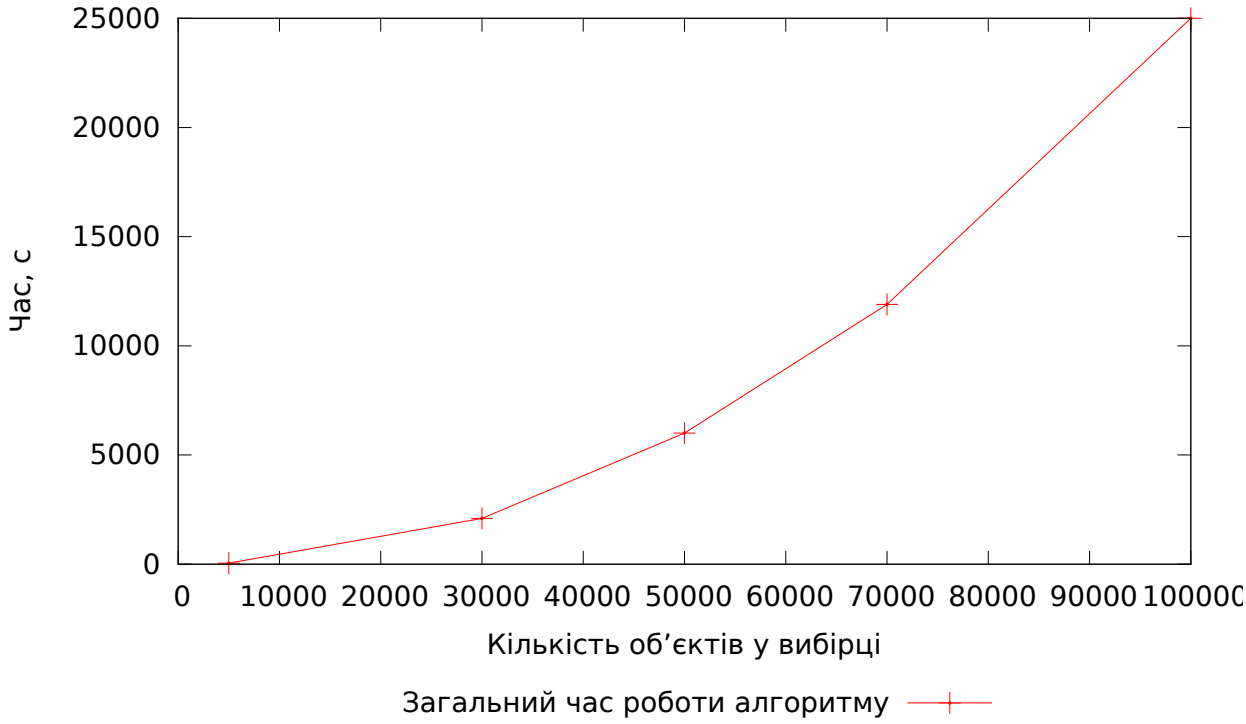

Рис. 3.9. Загальний час роботи neighbor-joining

3.4. Висновки до розділу 3

Для порівняння швидкодії алгоритмів, описаних у розділі 2, створено програмну реалізацію кожного із них. До реалізації застосовано інструменти профілювання та вимірювання часу виконання з метою виявлення найменш ефективних ділянок. В міру виявлення нефективні ділянки видалялись, що проілюстровано відповідними графіками покращення ефе-

ктивності алгоритмів. Вдалось досягти мінімальної теоретичної складності кожного алгоритму.

Кожен алгоритм перевірено на ідентичних наборах даних. Отримано числові результати, що відображають ефективність кожного алгоритму в різних ситуаціях.

Аналіз отриманих даних показав, що мікрооптимізації (зокрема, зменшення розрядності чисел та інші невеликі виправлення всередині програми) зазвичай мають лише мінімальний вплив на загальну ефективність програми. Макрооптимізації, у свою чергу, набагато сильніше впливають на швидкодію програми. До останніх можна віднести зміну структури даних, суттєві зміни у способах використання пам'яті, зміни алгоритмів, тощо.

3.5. Висновки

В даній роботі поставлено науково-практичну задачу розвитку алгоритмів кластеризації для опрацювання даних великих об'ємів.

Першим етапом її розв'язання є аналіз наукових напрацювань в цій темі та існуючих алгоритмів кластеризації. Розгляд структури алгоритмів кластеризації дозволив виявити потенційні проблеми при їх програмній реалізації та знайти спосіб здійснити її оптимальним чином.

Наступним кроком стало створення програми для генерування синтетичних наборів даних для перевірки роботи алгоритмів. Вона дала можливість перевіряти реалізацію алгоритмів на різних наборах даних та довести коректність їх роботи.

Останнім етапом дослідження стала програмна реалізація чотирьох алгоритмів кластеризації. Аналіз витрат ресурсів комп'ютера кожною реалізацією дав можливість виявити вузькі місця як власне реалізацій, так і алгоритмів. За допомогою цих даних вдалось внести у програму зміни, що суттєво прискорили її роботу в кожному із алгоритмів.

Аналіз результатів роботи програмного забезпечення показав високу ефективність алгоритму k-means. Кластеризація 100000 об'єктів 32-вимірного простору за допомогою цього алгоритму здійснюється за 16 секунд, а поліноміальна складність алгоритму дозволяє масштабувати його для даних ще більшого об'єму. Таким чином, якщо швидкодія є основним критерієм вибору алгоритму, доцільно обрати саме k-means.

Якщо ж в даних присутній шум чи відомо, що кластери мають форму, відмінну від сферичної, доцільно обрати DBSCAN. Цей алгоритм працює суттєво повільніше, аніж k-means (він потребує 12000 секунд для опрацювання масиву із 100000 об'єктів 32-вимірного простору), але здатен працювати із зашумленими даними та за відсутності інформації про кількість кластерів. При роботі із цим алгоритмом потрібно враховувати значний

вплив параметрів кластеризації на результат. Це стає його суттєвим недоліком — перед кластеризацією необхідно провести окремий аналіз вибірки для виявлення оптимальних значень параметрів.

Реалізація алгоритму neighbor-joining показала, що він є прийнятною заміною UPGMA для здійснення ієрархічної кластеризації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Arthur, D.* K-means has polynomial smoothed complexity / D. Arthur, B. Manthey, H. R. A. // Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009) / Ed. by D. Spielman. — Los Alamitos: IEEE Computer Society Press, 2009. — Pp. 405–414.
2. *Arthur, D.* How slow is the k-means method? / D. Arthur, S. Vassilvitskii // SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry. — New York, NY, USA: ACM, 2006. — Pp. 144–153.
3. *Davies, D. L.* A cluster separation measure / D. L. Davies, D. W. Bouldin // *IEEE Transactions on Pattern Analysis and Machine Intelligence.* — 1979. — Vol. 1, no. 2. — P. 224–227.
4. A density-based algorithm for discovering clusters in large spatial databases with noise / M. Ester, H.-P. Kriegel, S. Jörg, X. Xiaowei. — AAAI Press, 1996. — Pp. 226–231.
5. *Dunn, J. C.* Well separated clusters and optimal fuzzy partitions / J. C. Dunn // *Journal of Cybernetics.* — 1974. — Vol. 4. — Pp. 95–104.
6. *Eppstein, D.* Fast hierarchical clustering and other applications of dynamic closest pairs. / D. Eppstein // *J. Exp. Algorithmics.* — 2000. — no. 5. — Pp. 1–23.
7. *Hamerly, G.* Alternatives to the k-means algorithm that find better clusterings / G. Hamerly, C. Elkan // CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management. — New York, NY, USA: ACM, 2002. — Pp. 600–607.
8. *MacQueen, J. B.* Some methods for classification and analysis of multivariate observations / J. B. MacQueen // *Proceedings of 5th Berkeley Sympo-*

sium on Mathematical Statistics and Probability. — 1967. — P. 281–297.

9. Np-hardness of euclidean sum-of-squares clustering / D. Aloise, A. Deshpande, P. Hansen, P. Popat // *Machine Learning.* — 2009. — no. 75. — Pp. 245–249.
10. Performance evaluation of main-memory r-tree variants / S. Hwang, K. Kwon, S. K. Cha, B. S. Lee // SSTD. — 2003. — Pp. 10–27.
11. *Rosenblatt, F.* The perceptron – a perceiving and recognizing automaton: Tech. Rep. 85-460-1 / F. Rosenblatt: Cornell Aeronautical Laboratory, 1957.
12. *Saitou, N.* The neighbor-joining method: a new method for reconstructing phylogenetic trees. / N. Saitou, M. Nei // *Molecular Biology and Evolution.* — 1987. — Vol. 4, no. 4. — Pp. 406–425.
13. *Socal, R.* Principles of Numerical Taxonomy / R. Socal, P. Sneath. — San Francisco: W.H. Freeman, 1963.
14. *Sokal, R.* A statistical method for evaluating systematic relationships / R. Sokal, C. Michener // *University of Kansas Science Bulletin.* — 1958. — Vol. 38. — Pp. 1409–1438.
15. *Spielman, D.* Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time / D. Spielman, S. H. Teng // STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing. — New York, NY, USA: ACM, 2001. — Pp. 296–305.
16. *Steinhaus, H.* Sur la division des corps matériels en parties / H. Steinhaus // *Bulletin of the Polish academy of Sciences.* — 1956. — Vol. 3, no. 4. — Pp. 801–804.
17. *Tryon, R. C.* Cluster Analysis / R. C. Tryon. — Ann Arbor: Edwards Brothers, 1939.
18. *Загоруйко.* Прикладные методы анализа данных и знаний / Загоруйко. — Издательство института математики, Новосибирск, 1999.

19. *Терентьев П. В.* Метод корреляционных плеяд / Терентьев П. В. — Вестник Ленинградского университета, 1959.